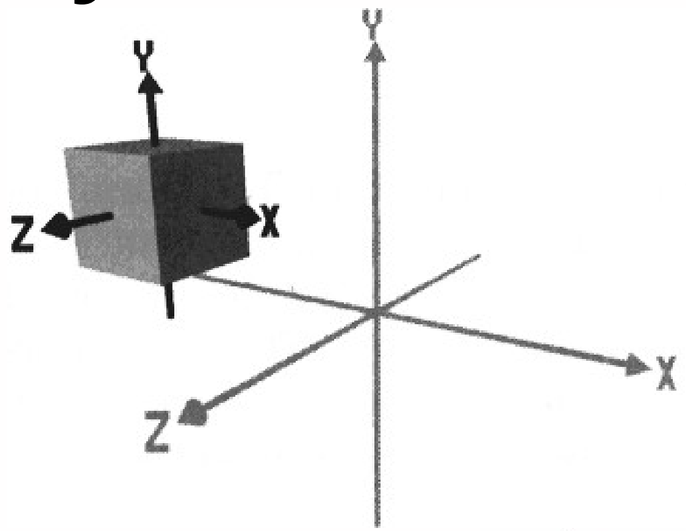
Ollscoil na Gaillimhe
University of Galway

# CT336/CT404 Graphics & Image Processing

4th year B.Sc. (CS&I.T.).
2nd year M.Sc. (Software Development & Design)
1st year M.Sc. (Biomedical Engineering)
Visiting students

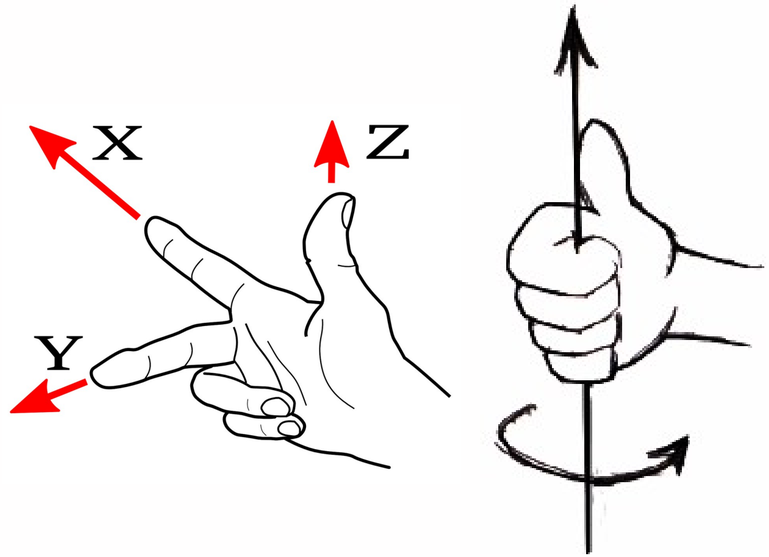
University *of* Galway.ie

# Lecture 3: Animation & Interactivity




- Last Lecture
  - 3D Coordinate Systems, Projections & Transformations
  - Introduction to Three.js
  - Three.js Examples: Primitives and Geometry, Nested Coordinates, Transformations

- Today
  - Animation & Interactivity both in Canvas 2D and Three.js
  - Shading, Materials & Lighting in Three.js
  - **Lots of Examples**



# Animation & Interactivity in Canvas 2D


Ollscoil na Gaillimhe
University of Galway

1. Handling the keyboard
   - Recognise keypresses and update graphics in response

2. Handling the mouse
   - Recognise mouse click/drag and update graphics in response

3. Time-based animation
   - Update graphics irrespective of user's actions => much better for any kind of multimedia/animation/games

# 1. Keyboard handling (Canvas/JavaScript)

Ollscoil na Gaillimhe
University of Galway

canvasWithKeyboardExample.html

```
<html>
 <head>
  <script>
function attachEvents() {
      document.onkeypress = function(event) {
            var xoffset=10*parseInt(String.fromCharCode(event.keyCode || event.charCode));
            draw(xoffset);
      }
}
function draw(xoffset) {

  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

    // remove previous translation if any
  context.save();
      // over-write previous content, with a white rectangle
  context.fillStyle="#FFFFFF";
  context.fillRect(0,0,300,300);
    // translate based on numerical keypress
  context.translate(xoffset,0);
    // purple rectangle
  context.fillStyle="#CC00FF";
  context.fillRect(0,0,50,50);
  context.restore();

}
  </script>
 </head>
 <body onload="attachEvents();">
   <canvas id="canvas" width="300" height="300"></canvas>
 </body>
</html>
```

# 1. Mouse handling (Canvas/JavaScript)

Ollscoil na Gaillimhe
University of Galway

```
<html>
 <head>
  <script>

var isMouseDown=false;

function attachEvents() {
        document.onmousedown = function(event) {
                isMouseDown=true;
                draw(event.clientX, event.clientY);
        }
        document.onmouseup = function(event) {
                isMouseDown=false;
        }
        document.onmousemove = function(event) {
                if ( isMouseDown ) {
                        draw(event.clientX,
event.clientY);
                }
        }
}
```

```
function draw(xoffset,yoffset) {

  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

    // remove previous translation if any
  context.save();
          // over-write previous content, with a grey rectangle
  context.fillStyle="#DDDDDD";
  context.fillRect(0,0,600,600);
    // translate based on position of mouseclick/drag
  context.translate(xoffset,yoffset);
    // purple rectangle
  context.fillStyle="#CC00FF";
  context.fillRect(-25,-25,50,50); // centred on coord system
  context.restore();

}
  </script>
 </head>

 <body onload="attachEvents(); draw(0,0);">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

canvasWithMouseExample.html

# 3. Time-based animation using *window.setTimeout* (repaints at pre-defined intervals))

Ollscoil na Gaillimhe
University of Galway

canvasAnimationExample1.html

```
<html>
 <head>
  <script>

var x=0, y=0;
var dx=4, dy=5;

function draw() {

  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

    // remove previous translation if any
  context.save();
          // over-write previous content, with a grey rectangle
  context.fillStyle="#DDDDDD";
  context.fillRect(0,0,600,600);

  // perform movement, and translate to position
   x+=dx;
   y+=dy;
   if (x<=0)
            dx=4;
   else if (x>=550)
     dx=-4;
   if (y<=0)
     dy=5;
   else if (y>=550)
     dy=-5;
   context.translate(x,y);
     // purple rectangle
   context.fillStyle="#CC00FF";
   context.fillRect(0,0,50,50);
   context.restore();

     // do it all again in 1/30th of a second
   window.setTimeout(draw, 1000/30);
}
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

# 3. Time-based animation (improved smoothness using *window.requestAnimationFrame* (called at every window repaint/refresh))

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

canvasAnimationExample1_withSmoother Animation.html

```
<html>
 <head>
  <script>
var x=0, y=0;
var dx=4, dy=5;
var now = Date.now();

function draw() {
  // do it all again in 1/60th of a second
  window.requestAnimationFrame(draw);

  var elapsedMs = Date.now() - now;
  now = Date.now();

  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

    // remove previous translation if any
  context.save();
  // over-write previous content, with a grey rectangle
  context.fillStyle="#DDDDDD";
  context.fillRect(0,0,600,600);
    // perform movement, and translate to position
  x+=dx*elapsedMs/16.7;
  y+=dy*elapsedMs/16.7;
  if (x<=0)
   dx=4;
  else if (x>=550)
    dx=-4;
  if (y<=0)
    dy=5;
  else if (y>=550)
    dy=-5;

  context.translate(x,y);
    // purple rectangle
  context.fillStyle="#CC00FF";
  context.fillRect(0,0,50,50);
  context.restore();
}
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

# Combined Mouse Clicks & Animation

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

canvasAnimationExample2.html

```
<html>
 <head>
  <script>
var boxes = [];

function attachEvents() {
        document.onmousedown = function(event) {
                // adds a new box at the mouse position
                // step 1: find a spare index in the sparse array 'boxes'
                var idx=Math.floor(Math.random()*1000);
                while (typeof boxes[idx]!="undefined")
                        idx=Math.floor(Math.random()*1000);
                // step 2: create a new box object and add to the array
                // setting up its data properties
                boxes[idx] = {};
                boxes[idx].x = event.clientX;
                boxes[idx].y = event.clientY;
                var r = Math.floor(Math.random()*256);
                var g = Math.floor(Math.random()*256);
                var b = Math.floor(Math.random()*256);
                boxes[idx].colr = "rgb("+r+","+g+","+b+")";
                boxes[idx].dy = Math.floor(1+Math.random()*8);
        }
}

function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

        // over-write previous content, with a grey rectangle
  context.fillStyle="#DDDDDD";
  context.fillRect(0,0,600,600);

   // iterate thru the objects in our sparse array
   // the for..in construct obtains *indices* rather
than *data values*
  for (var idx in boxes) {
                var y=boxes[idx].y+boxes[idx].dy; //
animate box downwards
                if (y<600) {
                        context.save();
                        boxes[idx].y=y;
                        context.translate(boxes[idx].x,
y);

context.fillStyle=boxes[idx].colr;
                        context.fillRect(0,0,20,20);
                        context.restore();
                }
                else
                        delete boxes[idx]; // box has
passed offscreen so delete it from array
  }

    // do it all again in 1/30th of a second
  window.setTimeout(draw, 1000/30);
}
  </script>
 </head>

<body onload="attachEvents(); draw();">
   <canvas id="canvas" width="600"
height="600"></canvas>
 </body>
</html>
```

e.g. "rgb(200,130,120)"

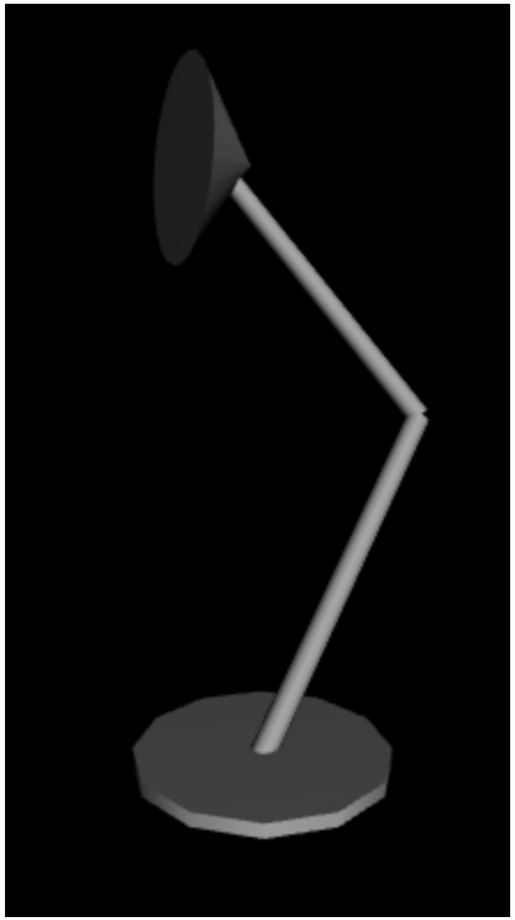
# Three.js: Animation & Interaction

- Raycasting is a feature offered by 3D graphics APIs
- Raycasting computes a ray from a start position in a specified direction, and identifies the geometry that this ray hits

```
 renderer = new THREE.WebGLRenderer({ canvas: c, antialias: true });
```

- Illustrates the use of raycasting/picking and rotation/translation based on mouse selection and mouse movement

- Also illustrates how nested coordinate systems have been used to make the lamp parts behave correctly



Threejs-20-controllable-desk-lamp.html

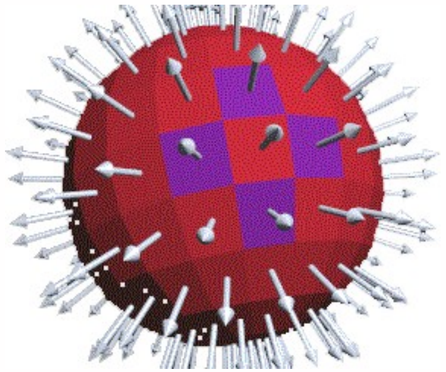
# Shading Algorithms

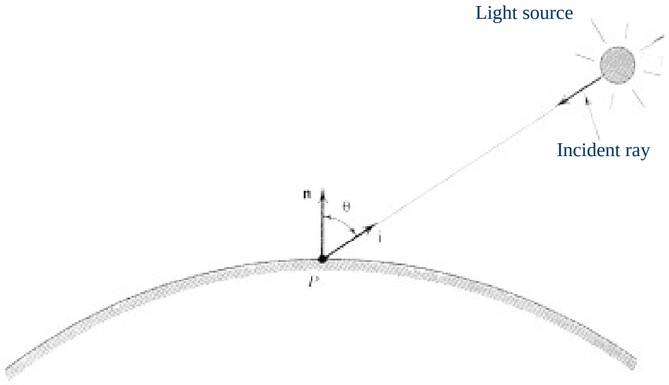
- The colour at any pixel on a polygon is determined by:
  - Characteristics (including colour) of the surface itself
  - Information about the light sources (ambient, directional – parallel or point, spot) and their positions relative to the surface
  - *Diffuse and specular reflections*

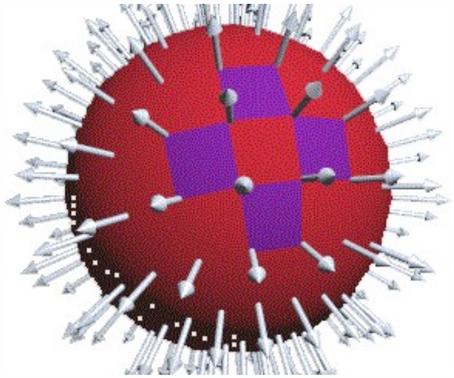
- Classic shading algorithms:
  - Flat shading
  - Smooth Shading (Gourard)
  - Normal Interpolating Shading (Phong)

Flat Gourard Phong

# Shading Algorithms



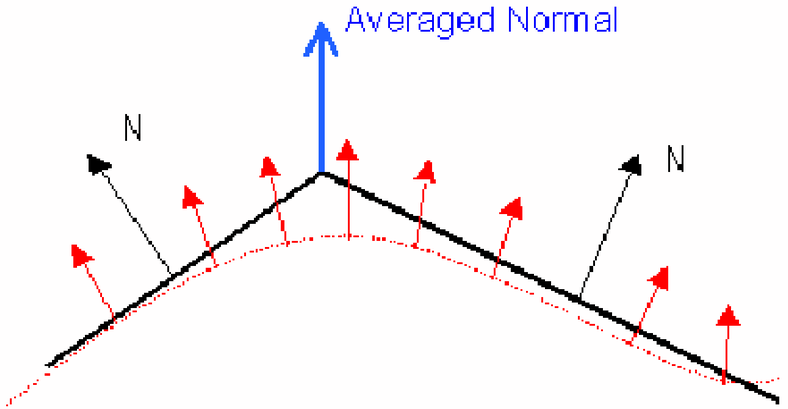


- Flat Shading
  - Calculates and applies directly the shade of each surface
  - Shade of surface calculated via the Cosine of the angle of incident ray to surface normal
  - A surface normal is a vector perpendicular to the surface

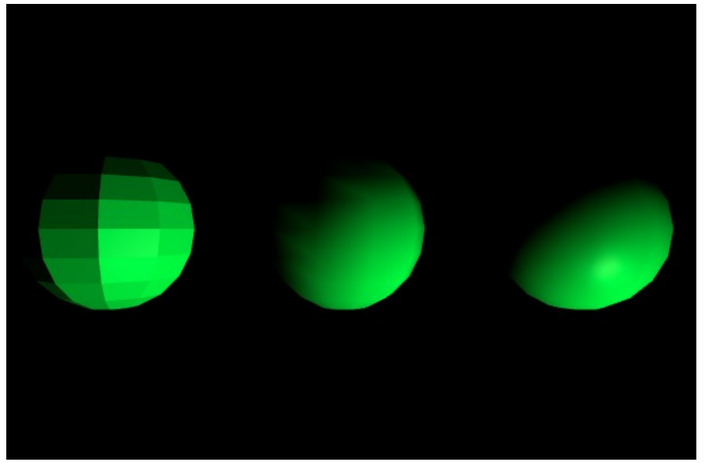
- Smooth (Gourard) Shading
  - calculates the shade at each vertex, and interpolates (smoothes) these shades across surfaces
  - Vertex normals are calculated by averaging normals of connected faces
  - Interpolation often carried out in graphics hardware (i.e. fast)

# Shading Algorithms




Black lines: actual edges
Red line: apparent edges wrt lighting

Example: Threejs-21-shading-algs.html



- Normal Interpolating (Phong) Shading

- Calculates the normal at each vertex, and interpolates these normals across the surfaces
- the light, and therefore shade, at each pixel is individually calculated from its unique surface normal

- Three.js Materials define how objects will look in the scene (https://threejs.org/manual/#en/materials)
- Shading models:
  - None = MeshBasicMaterial
  - Flat = MeshPhongMaterial with flatShading=true
  - Gourard = MeshLambertMaterial

# Three.js/WebGL Lighting

Ollscoil na Gaillimhe
University of Galway

- Six different types of lights are available:

  - **Point lights** - rays emanate in all directions from a 3D point source, (e.g. a lightbulb)
  - **Directional lights** - rays emanate in one direction only from infinitely far away (like the sun)
  - **Spotlights** - project a cone of light from a 3D point source, aimed at a specific target point.
  - **Ambient lights** – simulate in a simplified way the lighting of an entire scene due to complex light/surface interactions; lights up everything regardless of position or occlusion
  - **Hemisphere lights** – ambient lights that affect the ’ceiling’ or ‘floor’ hemisphere of objects rather than affecting them in entirety
  - **RectAreaLights** – emit rectangular areas of light (e.g. fluorescent light strip)

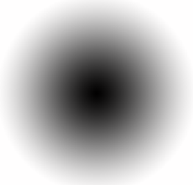
  - Example: Threejs-22-lights-examples.html

This example illustrates several types of light, and lets you turn them on and off via an HTML interface

# Three.js: Shadows



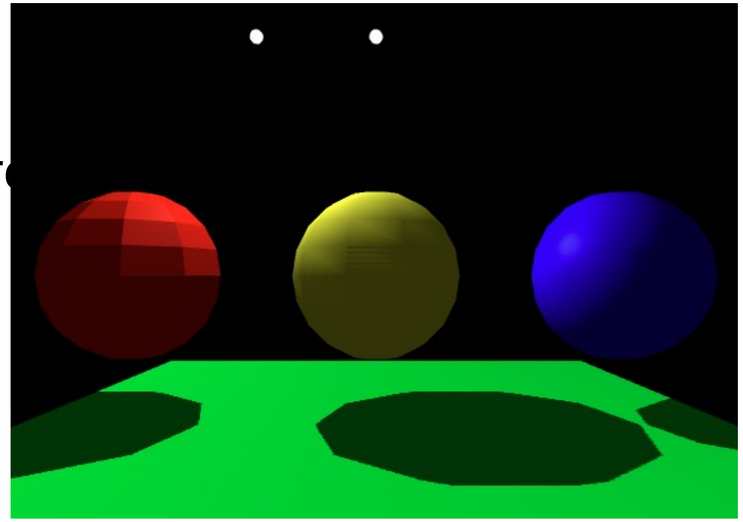
- Threejs supports programming of shadows, though they are expensive to use
- The scene is redrawn for each shadow-casting light, and finally composed from all the results
- Games sometimes use fake ‘blob shadows’ instead of proper shadows, or else only let one light cast shadows
- Resource: https://threejs.org/manual/#en/shadows



Blob shadow

- Example: Threejs-25-lights-and-shadows.html

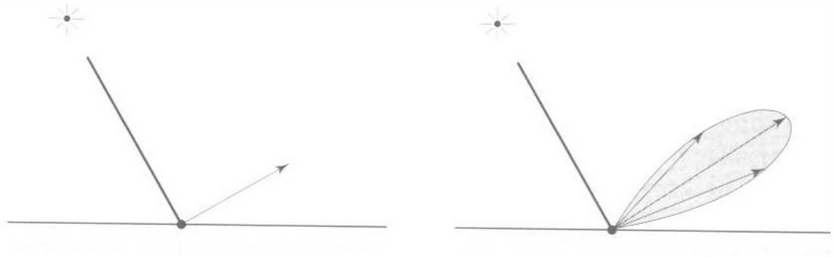
- For Three.js textures: https://threejs.org/manual/#en/textur



# Three.js Reflectivity of Materials

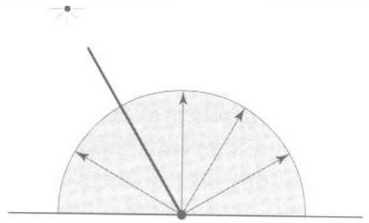


- Colour Settings:
  - Diffuse colour: Defined by the colour setting
  - Specular colour: Colour of specular highlights (Phong only)
  - Shininess: Strength of specular highlights (Phong only)
  - Emissive colour: Colour is not affected by lighting – cf MeshBasicMaterial



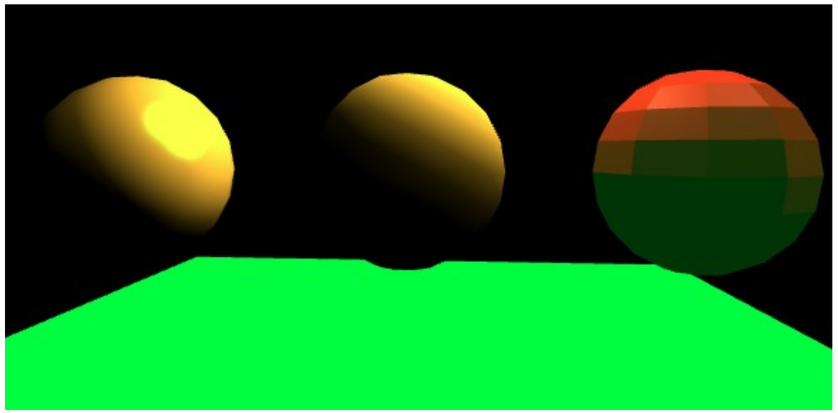
Perfect Specular Reflection

Imperfect Specular Reflection



Perfect Diffuse Reflection

Example: Threejs-23-materials-examples.html



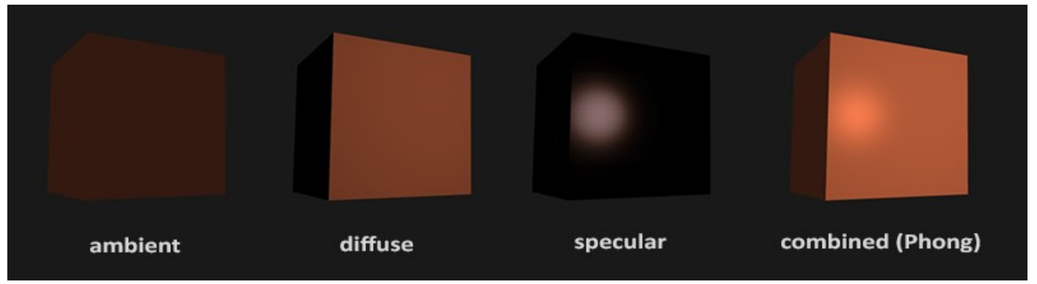



Image source: https://learnopengl.com/Lighting/Basic-Lighting

# Summary of Graphics Syllabus covered


Ollscoil na Gaillimhe
University of Galway

- What are images? Image encoding schemes
- Graphics Pipeline, Libraries, Hardware
- 2D vector & raster graphics
- 2D Transformations
- 3D Coordinate Systems, Projections & Transformations
- 3D Primitives and Geometry
- Animation & Interactivity both in 2D and 3D
- 3D Shading, Materials, Lighting & Shadows
- HTML5/Canvas for applied 2D graphics
- Three.js for applied 3D graphics

- ***Assignment: Get creative!***

- ***NEXT: Here comes the exciting world of image processing and computer vision!***

Ollscoil na Gaillimhe
University of Galway

# Thank *you*